

```
02:57:49.920864 IP 192.168.56.101.36286 > 192.168.56.103.sip-tls: Flags [P.], seq 34904:35597, ack 35219, win 1205, options [nop,nop,TS val 1514174486 ecr 1572301], length 693
```

```
0x0000: 4500 02e9 dda3 4000 4006 684e c0a8 3865 E.....@.@.hN..8e
0x0010: c0a8 3867 8dbe 13c5 49bf 2142 37dd f161 ..8g....I.!B7..a
0x0020: 8018 04b5 d869 0000 0101 080a 5a40 7816 .....i.....Z@x.
```

## >Attacking encrypted VoIP protocols

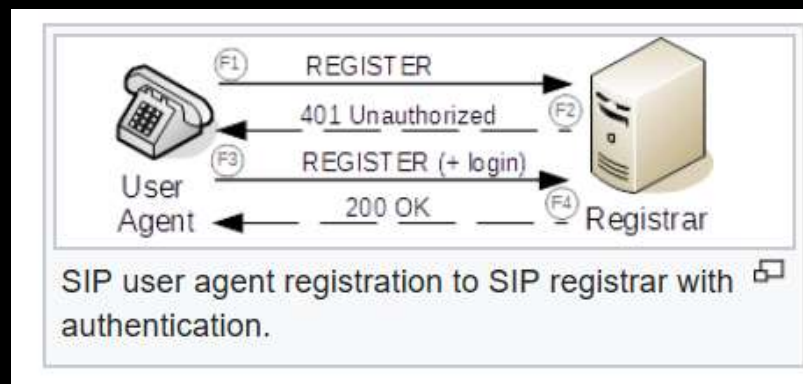
```
0x0060: dbef 4a64 5737 b4e7 a3c6 1d5a 6c84 4fc3 ..JdW7.....Zl.O.
0x0070: 8daf 0422 31fc 8177 ddab 1835 5256 e08d ..."1..w...5RV..
0x0080: 50a0 3aa7 94ca 1c48 bd7e 6ed7 f907 ba95 P.:....H.~n.....
0x0090: 791e f63a 9bb9 f084 9e25 a9db c01f 1da2 y.:.....%.
0x00a0: 0814 b053 3587 6e07 5903 21e5 96be d44b ...S5.n.Y.!....K
0x00b0: 56cb 1662 8a0c 7d2b f952 5933 ae51 71b0 V..b..}+.RY3.Qq.
0x00c0: ef53 ad93 1653 f6bf eb04 19f1 55a6 abc5 .S...S.....U...
0x00d0: 5d90 d49f cf19 1cc9 053c 79ad e019 ab35 ].....<y....5
0x00e0: a9a3 abbe d1a4 c550 1211 2abb 9977 9b80 .....P..*..w..
0x00f0: aa7b f38f f2da 535f 02a2 3020 2d67 0ae9 .{....S...0.-a..
```

## Biography

1. Name>Ivica [Eeveetsa] Stipovic
2. Work>Ward Solutions, Dublin, Ireland
3. Job>Information Security Consultant
4. Contact> [Ivica.Stipovic@ward.ie](mailto:Ivica.Stipovic@ward.ie)
5. >EOF

## What is VoIP protocol ?

- VOIP stands for (Voice Over IP) - Voice /Video/messaging that uses IP-based transport protocols for transmission
- What is SIP?
  - SIP stands for Session Initiation Protocol (it is a voice control protocol), developed by IETF
- SIP is one of the predominant VOIP control protocols



Source: [https://en.wikipedia.org/wiki/Session\\_Initiation\\_Protocol](https://en.wikipedia.org/wiki/Session_Initiation_Protocol)

## SIP, MGCP, H.323, XMPP...

There are other VoIP protocols as well

- MGCP Media Gateway Control Protocol (MGCP), connection management for media gateways
- H.323 - one of the first VoIP call signaling and control protocols that found widespread implementation
- XMPP - Extensible Messaging and Presence Protocol , instant messaging, presence information, and contact list maintenance
- Skype protocol, proprietary Internet telephony protocol suite based on peer-to-peer architecture

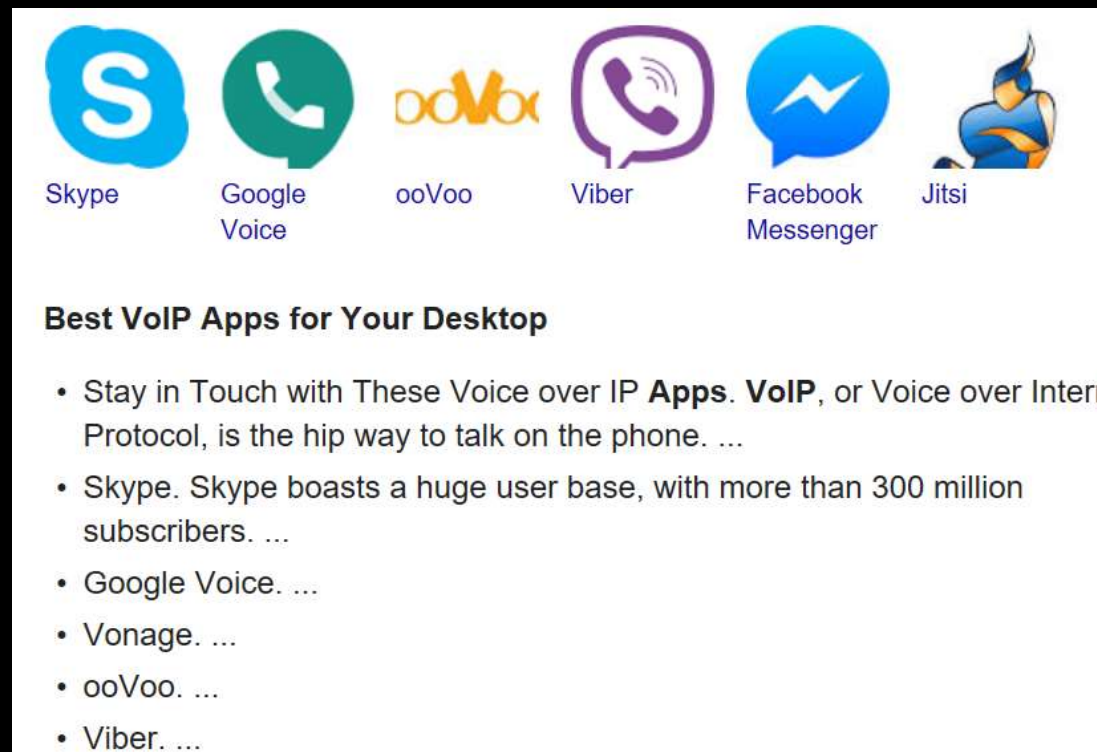
...etc

# What applications are using VoIP?

This is a small snapshot of the most popular VoIP applications.

Not all of them use SIP, though:  
Skype uses proprietary protocol

Viber does use SIP, so as Blink,  
Cisco IP Communication, Jitsi,  
Bria, Empathy, Linphone, X-Lite,  
Zoiper, Yahoo! Messenger...



## Let's talk about SIP

- SIP by default runs as a UDP service over the port 5060
- By default, it is not encrypted
- Full description of SIP protocol is given in the <https://tools.ietf.org/html/rfc3261>
- SIP structure is very similar to HTTP session structure (both request and response paradigm)

# Structure of a SIP session

Similar to HTTP

Request (REGISTER vs GET)

Response (Unauthorised  
Return code 401)

Request (REGISTER vs GET)

```
REGISTER sip:192.168.0.116:5061 SIP/2.0
Call-ID: 2b0a036d62188707e8c1fa3f99d442eb@0:0:0:0:0:0:0
CSeq: 1 REGISTER
From: "demo-103" <sip:demo-103@192.168.0.116>;tag=fe23e8cf
To: "demo-103" <sip:demo-103@192.168.0.116>
Via: SIP/2.0/TLS 192.168.0.164:3565;branch=z9hG4bK-373232-13573379b4dc2c69a
Max-Forwards: 70
User-Agent: Jitsi2.10.5550Windows 10
Expires: 600
Contact: "demo-103" <sip:demo-103@192.168.0.164:3565;transport=tls;register>
Content-Length: 0
```

```
SIP/2.0 401 Unauthorized
Via: SIP/2.0/TLS
192.168.0.164:3565;branch=z9hG4bK-373232-13573379b4dc2c69aa6994b76a28d8af;r
From: "demo-103" <sip:demo-103@192.168.0.116>;tag=fe23e8cf
To: "demo-103" <sip:demo-103@192.168.0.116>;tag=as174e6496
Call-ID: 2b0a036d62188707e8c1fa3f99d442eb@0:0:0:0:0:0:0
CSeq: 1 REGISTER
Server: Asterisk PBX 15.7.2
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY, INFO, P
Supported: replaces, timer
WWW-Authenticate: Digest algorithm=MD5, realm="asterisk", nonce="08e37204"
Content-Length: 0
```

```
REGISTER sip:192.168.0.116:5061 SIP/2.0
Call-ID: 2b0a036d62188707e8c1fa3f99d442eb@0:0:0:0:0:0:0
CSeq: 2 REGISTER
From: "demo-103" <sip:demo-103@192.168.0.116>;tag=fe23e8cf
To: "demo-103" <sip:demo-103@192.168.0.116>
Max-Forwards: 70
User-Agent: Jitsi2.10.5550Windows 10
```



## Attributes of a SIP session

We will focus on attributes required to facilitate password attacks. These are: algorithm, method, username, realm, uri, response, nonce, {nc}, {cnonce} and {qop}

```
SUBSCRIBE sip:demo-103@192.168.0.116 SIP/2.0
Call-ID: 7f8aa645e0eaa29432637916eacf4fac@0:0:0:0:0:0
CSeq: 2 SUBSCRIBE
From: "demo-103" <sip:demo-103@192.168.0.116>;tag=27f22122
To: "demo-103" <sip:demo-103@192.168.0.116>
Max-Forwards: 70
Contact: "demo-103" <sip:demo-103@192.168.0.164:3565;transport=tls;registering_acc=192_168_0_116>
User-Agent: Jitsi2.10.5550Windows 10
Event: presence.winfo
Accept: application/watcherinfo+xml
Expires: 3600
Via: SIP/2.0/TLS 192.168.0.164:3565;branch=z9hG4bK-373232-b62e3236d4729306c94af25ea7b34831
Authorization: Digest
username="demo-103",realm="asterisk",nonce="5ef0fb06",uri="sip:demo-103@192.168.0.116",response="a35726f18b7e4d8e62648dc299f28241",algorithm=MD5
Content-Length: 0
```



## Why TLS/encryption?

- wrapping SIP into TLS makes it more secure (HTTP vs HTTPS, POP3 vs POP3S, LDAP vs LDAPS etc.)
- interception of encrypted SIP will show obfuscated application layer payload
- only ip/tcp/udp header level information intelligible

# Why TLS/encryption?

-challenges to pentesters that TLS presents – we want the application layer

Lower OSI layers



Application layer  
obfuscated



255	38.962338	192.168.0.164	192.168.0.116	TPKT	215	Continuation
256	38.963635	192.168.0.116	192.168.0.164	TCP	60	5061 → 2945 [ACK]
257	38.966307	192.168.0.116	192.168.0.164	TCP	60	5061 → 2945 [ACK]
258	38.966902	192.168.0.116	192.168.0.164	TCP	60	5061 → 2945 [ACK]
265	39.041986	192.168.0.116	192.168.0.164	TPKT	636	Continuation
267	39.081645	192.168.0.164	192.168.0.116	TCP	54	2945 → 5061 [ACK]
268	39.157116	192.168.0.116	192.168.0.164	TPKT	652	Continuation
269	39.196799	192.168.0.164	192.168.0.116	TCP	54	2945 → 5061 [ACK]
294	40.865386	192.168.0.164	192.168.0.116	TPKT	443	Continuation
304	41.068180	192.168.0.116	192.168.0.164	TCP	60	5061 → 2945 [ACK]

Frame 294: 443 bytes on wire (3544 bits), 443 bytes captured (3544 bits) on interface 0
Ethernet II, Src: IntelCor_48:77:f9 (3c:6a:a7:48:77:f9), Dst: IntelCor_80:65:d3 (18:56:80:80:65:d3)
Internet Protocol Version 4, Src: 192.168.0.164, Dst: 192.168.0.116
Transmission Control Protocol, Src Port: 2945, Dst Port: 5061, Seq: 13079, Ack: 6387, Len: 385
TPKT - ISO on TCP - RFC1006
Continuation data: 1703030180ab648c3bc196077d8836337c44874b13a6a99c...

0	01 00 f4 9a 00 00	17 03 03 01 80 ab 64 8c 3b c1	.....d;.
0	96 07 7d 88 36 33 7c 44	87 4b 13 a6 a9 9c 56 f1	..}.63 D.K...V.
0	34 9d 9d ef 1b de 5d bf	bc dc 55 d3 2f aa 02 cd	4.....].U./...

## SIP –two aspects of attacks

1<sup>st</sup> aspect : interception + decryption

- for unencrypted SIP sessions, one focuses only on interception
- in our case, we need to do both

2<sup>nd</sup> aspect : SIP password cracking

- for unencrypted SIP sessions, off-the-shelf tools available in Kali
- in our case, we need to develop either:
  - manual preparation of a file for sipcrack
  - our own tool to streamline the cracking (<= chosen approach)

# Attacking plaintext SIP passwords

-Kali with sipdump and sipcrack

-sipdump takes a pcap of a SIP session as input and generates a text file output for sipcrack

-sipcrack takes the text output from sipdump and performs password dictionary attack

```
root@kali:/projekti/mitm-relay# sipdump -h

SIPdump 0.2
-----

Usage: sipdump [OPTIONS] <dump file>

    <dump file>    = file where captured logins will be written to

Options:
  -i <interface> = interface to listen on
  -p <file>       = use pcap data file
  -m              = enter login data manually
  -f "<filter>"    = set libpcap filter

* Invalid arguments
root@kali:/projekti/mitm-relay# sipcrack -h

SIPcrack 0.2
-----

Usage: sipcrack [OPTIONS] [ -s | -w <wordlist> ] <dump file>

    <dump file>    = file containing logins sniffed by SIPdump

Options:
  -s              = use stdin for passwords
  -w wordlist      = file containing all passwords to try
  -p num          = print cracking process every n passwords (for -w)
                   (ATTENTION: slows down heavily)

* Invalid arguments
root@kali:/projekti/mitm-relay#
```

## Solution design-1st part (interception and decryption)

An idea of MITM occurred as one plausible attack vector

This is what we need to achieve:

- build a mechanism capable of intercepting and decrypting the TLS wrapped session
- search for some kind of protocol-agnostic proxy capable of decrypting TLS
- forward the traffic from this protocol-agnostic proxy to Burp so we can play with packets

## Solution design-1st part (interception and decryption)

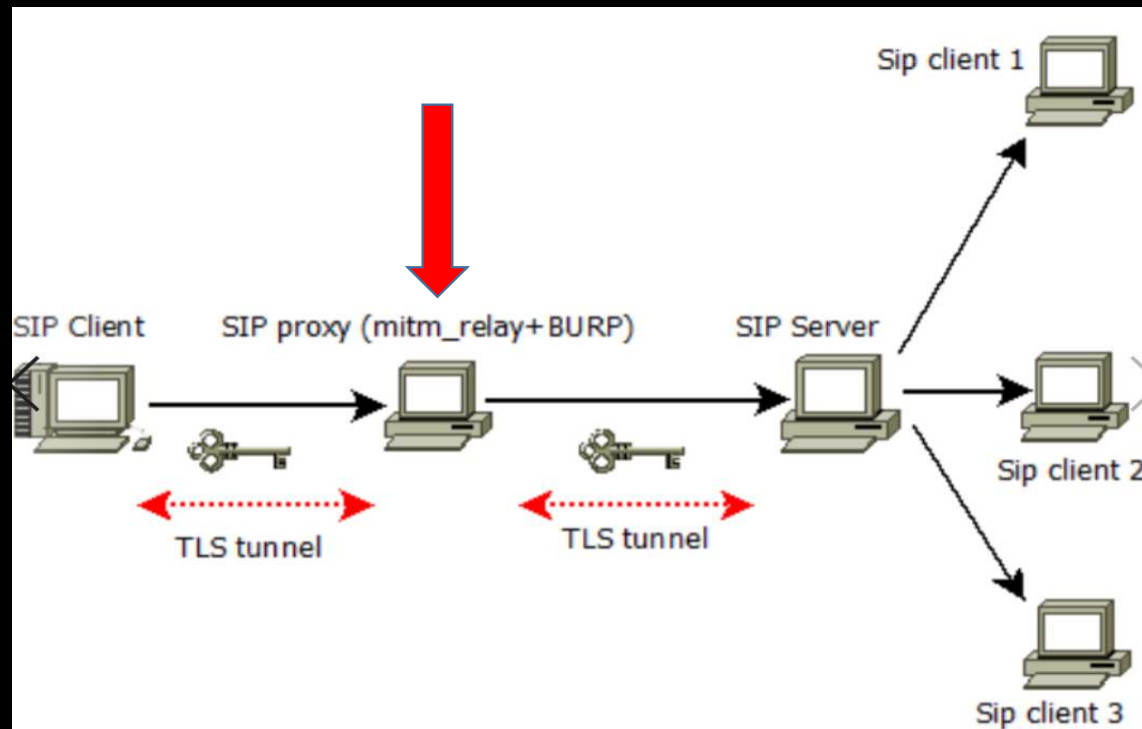
- Burpsuite does that job No. 1, but only for HTTP(S) – it does not speak SIP or any other non-HTTP(S) protocols for that matter
- The Solution: mitm\_relay.py ([https://github.com/jrmdev/mitm\\_relay](https://github.com/jrmdev/mitm_relay))



## Solution design-1st part (interception and decryption)

-Topology design developed as below

- Key component: mitm\_relay.py + Burpsuite running as SIP proxy on Kali
- you can have them on separate VMs too, also , no particular need for Kali – any linux will do, I guess



## Chaining mitm\_relay.py with Burp

-l 0.0.0.0 mitm\_relay.py listens on all interfaces

-p 127.0.0.1:8080 Burp runs on localhost, port 8080

-r tcp:5061:192.168.0.101:5061 relay all traffic on tcp 5061 port to final SIP server running on 192.168.0.101, port 5061

-c mitm.pem digital certificate for mitm\_relay.py

-k privatemitm.key private key for mitm\_relay.py

```
root@kali:~/projekti/VOIP# !174
python /opt/mitm_relay.py -l 0.0.0.0 -p 127.0.0.1:8080 -r tcp:5061:192.168.0.101:5061 -c mitm.pem -k privatemitm.
key 10.1.49999 POST /CLIENT REQUEST to 192.168.0.101/5... ✓ 200 360
[+] Client cert/key not provided ✓ 200 168
[+] Webserver listening on ('127.0.0.1', 49999) ✓ 200 612 text
[+] Relay listening on tcp 5061 -> 192.168.0.101:5061 ✓ 200 689 script
[+] New client: ('192.168.56.1', 61984) -> 192.168.0.101:5061 ✓ 200 777 text
Wrapping sockets ✓ 200 755 script
C >> S [ 192.168.56.1:61984 >> 192.168.0.101:5061 ] [ Fri 05 Apr 17:55:44 ] [ 267 ]
00000000: 16 03 03 01 06 10 00 01 02 01 00 30 8f ec a7 03 |.....0....| script
00000010: af 4a a4 ff e5 c4 6c 32 a2 a3 d7 4d 7d b0 14 ac |.J....l2...M)...| script
00000020: 5a 5c 1a be c1 68 27 93 df ad ca 9d 74 5b 1c 78 |Z...h'.....t.X| text
00000030: 7c 71 2f 60 0b b0 c4 29 61 1d ba f2 6f df 3e 0e ||q/'...a...o.>..| text
00000040: 87 f0 46 29 8f 40 08 da 05 70 44 ba b5 22 17 35 |..F).@...pD..."5| text
00000050: 3f f1 e3 33 0c e8 0a df d9 f1 30 c1 2b ba f8 91 |?...3.....0.+...| script
00000060: 03 37 9e 91 c0 d9 4b ab 2e 7b e2 48 47 de 49 20 |.7...K...{.HG.I..| text
00000070: c8 7e 76 c8 4d e4 6f 62 8a 0d 61 54 97 e2 7b 51 |.~v.M.ob...aT...{0|
00000080: b8 d0 0e 08 be c7 6c 4a 89 b7 47 10 fd 61 b5 f2 |.....lJ..G..a...|
00000090: 74 ea 2d b7 3d 1a 09 bf 84 69 3c 8b 89 93 5a f3 |t.-.=...i<...Z..|
000000a0: f7 29 5b d2 1b a2 28 e7 f3 59 ae ba e3 3a f3 b4 |.)[(...Y.....|
000000b0: e0 3b ad 97 a1 8a 2c 7b 98 a2 11 bd f3 5b df 44 |.;.....{.....[.D|
000000c0: 3e 77 28 76 84 8c 6a 5b b9 89 c7 a4 ff e4 76 e6 |>w(v..j[.....v..|
000000d0: 01 ec 93 ce 13 5f 48 97 98 5b ae dc 0f 5b 3f 2f |....._H...[...[?/|
000000e0: 97 ae 9e a6 2a a9 35 ab ed ae 44 77 07 f4 ee 69 |....*.5...Dw...i|
000000f0: 54 a8 ed 4e e3 80 a1 d5 8c fa 09 de f1 38 ad d5 |T..N.....8...|
00000100: b4 67 cb 48 54 8f 2a 0e ec c5 07 |.g.HT.*....|
pidemo-102@192.168.56.103 SIP/2.0
390de85c9c1b142c7988b6ca950b0@0:0:0:0:0:0
CR18F
```

## Chaining mitm\_relay.py with Burp

- If we switch over to Burp, we can see decrypted SIP negotiation
- POST requests have embedded “/CLIENT\_REQUEST/to/<IP>” and “/SERVER\_RESPONSE/from/<IP>”
- that allows Burp to process SIP Sessions (it only transports stuff, does not really speak SIP)

The screenshot shows the Burp Suite interface. The top menu bar includes Target, Proxy, Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, and Project. Below the menu bar are tabs for Intercept, HTTP history, WebSockets history, and Options. The HTTP history tab is active, displaying a table of intercepted requests. The table has columns for #, Host, Method, URL, Params, Edited, and Size. The 14th entry is highlighted in orange.

#	Host	Method	URL	Params	Edited	Size
1	http://127.0.0.1:49999	POST	/CLIENT_REQUEST/to/192.168.0.101/5...	✓		2
2	http://127.0.0.1:49999	POST	/CLIENT_REQUEST/to/192.168.0.101/5...	✓		2
3	http://127.0.0.1:49999	POST	/CLIENT_REQUEST/to/192.168.0.101/5...	✓		2
4	http://127.0.0.1:49999	POST	/SERVER_RESPONSE/from/192.168.0.1...	✓		2
5	http://127.0.0.1:49999	POST	/CLIENT_REQUEST/to/192.168.0.101/5...	✓		2
6	http://127.0.0.1:49999	POST	/SERVER_RESPONSE/from/192.168.0.1...	✓		2
7	http://127.0.0.1:49999	POST	/SERVER_RESPONSE/from/192.168.0.1...	✓		2
8	http://127.0.0.1:49999	POST	/CLIENT_REQUEST/to/192.168.0.101/5...	✓		2
9	http://127.0.0.1:49999	POST	/SERVER_RESPONSE/from/192.168.0.1...	✓		2
10	http://127.0.0.1:49999	POST	/CLIENT_REQUEST/to/192.168.0.101/5...	✓		2
11	http://127.0.0.1:49999	POST	/SERVER_RESPONSE/from/192.168.0.1...	✓		2
12	http://127.0.0.1:49999	POST	/CLIENT_REQUEST/to/192.168.0.101/5...	✓		2
13	http://127.0.0.1:49999	POST	/CLIENT_REQUEST/to/192.168.0.101/5...	✓		2
14	http://127.0.0.1:49999	POST	/CLIENT_REQUEST/to/192.168.0.101/5...	✓		2
15	http://127.0.0.1:49999	POST	/SERVER_RESPONSE/from/192.168.0.1...	✓		2
16	http://127.0.0.1:49999	POST	/CLIENT_REQUEST/to/192.168.0.101/5...	✓		2

Below the table, the 'Request' tab is selected, showing the raw HTTP request details:

```
POST /CLIENT_REQUEST/to/192.168.0.101/5061 HTTP/1.1
Host: 127.0.0.1:49999
Accept-Encoding: gzip, deflate
X-Mitm_Relay-To: 192.168.0.101:5061
X-Mitm_Relay-From: 192.168.56.1:61987
Content-Length: 751
Connection: close

SUBSCRIBE sip:demo-102@192.168.56.103 SIP/2.0
Call-ID: 432390de85c9c1b142c7988b6ca050b0@0:0:0:0:0:0:0:0
CSeq: 2 SUBSCRIBE
From: "demo-102" <sip:demo-102@192.168.56.103>;tag=3937750d
To: "demo-102" <sip:demo-102@192.168.56.103>
Max-Forwards: 70
Contact: "demo-102" <sip:demo-102@192.168.56.1:61987;transport=tls;registering_acc=192_168_56_103>
User-Agent: Jitsi2.10.5550Windows 10
Event: message-summary
Accept: application/simple-message-summary
Expires: 3600
Via: SIP/2.0/TLS 192.168.56.1:61987;branch=z9hG4bK-3134-84296c6f16c0f022f365fd1c78e83979
```

## Chaining mitm\_relay.py with Burp

- Mechanics of the interception
- SIP client -> mitm\_relay.py -> Burpsuite -> SIP server -> SIP client 2
- NOTE: remember that now that we have Burp reading the SIP, several other attacks can be mounted :
  - send SIP request to Burp Repeater,
  - change call destinations,
  - brute force destination numbers,
  - change user agent fingerprint ,
  - inject some funky headers/establish covert channel attack,
  - spoof calling ID...
- Tampering in BURP is interesting, but out of scope of this research

## Solution design - 2nd part (coding new app)

- Why is new app required –what about sipdump and sipcrack?
  - sipdump will not work here –we have dumped the decrypted session into a non-pcap format
  - sipdump has no way of importing the private key to decrypt the captured wireshark pcap session
  - we can decrypt wireshark pcap, remember? – we possess the private key for mitm\_relay.py
- What is new app doing?
  - new app will process the mitm\_relay.py dump which is more-less text based file and will extract all authentication attributes and perform password dictionary-based attack

## Solution design - 2nd part (coding new app)

- Functional mapping of sipdump+sipcrack into sipcrack2
- Streamline mitm\_relay dump parsing with password cracking
- How do we crack digest algorithm?

-<https://www.ietf.org/rfc/rfc2069.txt> later ammended by RFC 2617

-{qop},{nc},{cnonce} increase the variability of hashing but will still not protect against our attack and all these attributes can be intercepted and decrypted



## Solution design - 2nd part (coding new app)

- Digest authentication algorithm
- If {qop},{cnonce} and {nc} are not defined (RFC2069) then  
     $H1 = \text{MD5}(\text{username}:\text{realm}:\text{password})$   
     $H2 = \text{MD5}(\text{method}:\text{uri})$   
     $\text{Response} = \text{MD5}(H1:\text{nonce}:H2)$

Else if {qop},{cnonce} and {nc } are defined (RFC2617) then

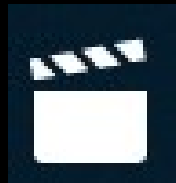
$\text{Response} = \text{MD5}(H1:\text{nonce}:\text{nc}:\text{cnonce}:\text{qop}:H2)$

<https://github.com/adenosine-phosphatase/sipcrack2>

## Final thoughts

- Development  
sipcrack2 is for now just a linux version, hope to release Windows version with CUDA/multithreading & parallel processing in a near future
- How realistic/difficult is the attack?  
The attack shown is relatively difficult to implement (ARP poisoning required to redirect the traffic from legitimate proxy to the attacker)
- Recommendations
  - use strong passwords
  - do not use self-signed certificates
  - use client side certificate in addition to server
- Risk rating proposed: Medium

## Demo



## Questions?

```
Telling INIT to go to single user mode.  
init: rc main process (2205) killed by TERM signal  
[root@centos-4 ~]# _
```

Shutting down